VISUAL BASIC

ขจร โรจนเมธินทร์

Visual BASIC : เขียนโปรแกรมให้มีประสิทธิภาพสูง

นำเสนอเทคนิคต่าง ๆ ที่จะนำไปใช้ในการเขียนโปรแกรม ซึ่งจะทำให้โปรแกรมที่เขียนทำงานได้มีประสิทธิภาพมากขึ้น

การเขียนโปรแกรมให้มีประสิทธิภาพนั้นประกอบด้วยกันหลายขั้นตอน โปรแกรมที่มีประสิทธิภาพนั้นผู้เขียนจะต้องนำเทคนิคต่าง ๆ ที่จะกล่าวถึงในบทความนี้เข้ามาประยุกต์ในแต่ละขั้นของการพัฒนาโปรแกรม การตัดสินว่าโปรแกรมใดมีประสิทธิภาพนั้นโดยทั่วไปก็อาจใช้ขนาดและความเร็วในการทำงานเป็นเครื่องตัดสิน แต่ข้อกำหนดทั้งสองนี้บางครั้งก็ไม่สามารถเกิดขึ้นพร้อมกันได้ นอกจากนี้การเขียนโปรแกรมให้ทำงานได้เร็วนั้นก็อาจมีผลให้โปรแกรมนั้นแก้ไขยากเมื่อมีข้อผิดพลาดขึ้นด้วย

บางครั้งผู้อ่านอาจต้องชั่งน้ำหนักกันระหว่างประโยชน์ที่ได้จากการเขียนโปรแกรมให้มีประสิทธิภาพสูงกับเวลาที่เสียไป เพราะการเขียนโปรแกรมในลักษณะนี้จะต้องใช้เวลามาก โดยเฉพาะการสังเกตการทำงานของโปรแกรม และที่สำคัญก็ยังไม่มีหลักเกณฑ์ตายตัวที่จะนำไปใช้ได้กับทุกโปรแกรม จุดที่ควรให้ความสนใจมากที่สุดก็คือ ส่วนการทำงานที่เป็นลูปของโปรแกรม ซึ่งจะกินเวลาในการทำงานมากที่สุด อย่างไรก็ตาม ผู้อ่านควรจะทราบว่าบางครั้งเราก็ไม่มีความจำเป็นจะต้องเขียนโปรแกรมให้มีประสิทธิภาพเสมอไป เช่น ในการเขียนโปรแกรมขนาดเล็กที่มีการทำงานไม่ซับซ้อนนัก เราก็ไม่จำเป็นต้องมาเสียเวลาปรับปรุงให้โปรแกรมนั้นมีประสิทธิภาพ หรือบางครั้งโปรแกรมที่เราเขียนขึ้นก็ไม่สามารถจะทำให้มีประสิทธิภาพสูงขึ้นอีกได้ เช่น โปรแกรมที่ต้องทำงานเกี่ยวข้องกับฮาร์ดดิสค์หรือติดต่อเพื่อเรียกข้อมูลผ่านเน็ตเวอร์ก ความเร็วของโปรแกรมจะถูกจำกัดด้วยประสิทธิภาพของฮาร์ดแวร์ที่ใช้งานอยู่

ในบทความนี้จะกล่าวถึงหลักการที่ผู้อ่านสามารถนำไปปรับปรุงโปรแกรม โดยแบ่งเป็นหัวข้อที่จะต้องพิจารณาทั้งหมด 5 เรื่องดังนี้

1. Real speed เป็นการเพิ่มความเร็วในส่วนของการทำงานของโปรแกรมโดยเฉพาะ

2. Display speed เป็นการเพิ่มความเร็วในส่วนที่ทำหน้าที่ระบายสี (Paint) หรือเพนท์จอภาพของโปรแกรม

3. Apparent speed เป็นการเพิ่มความเร็วในการปรากฏตัวของโปรแกรมเมื่อถูกเรียกมาทำงาน

4. Size in memory ลดขนาดการใช้หน่วยความจำของโปรแกรม

5. Size of graphics ลดการใช้กราฟิกต่าง ๆ ให้น้อยลง ซึ่งจะมีผลต่อการใช้หน่วยความจำ

ปรับปรุง Real speed

ปกติแล้วความเร็วเฉพาะส่วนการทำงานของโปรแกรมจริง ๆ มักไม่ค่อยมีผลมากนักถ้าโปรแกรมไม่ได้คำนวณอะไรมากมาย ผลลัพธ์ความเร็วของโปรแกรมมักจะขึ้นอยู่กับความเร็วของการ์ดแสดงผล เน็ตเวอร์ก ฮาร์ดดิสค์มากกว่า สาเหตุส่วนใหญ่ที่ทำให้ฟอร์มโหลดช้า อาจมีเนื่องมาจากฟอร์มนั้นมีคอนโทรลหรือกราฟิกอยู่หลายอันมากกว่าเกิดจากโค้ดคำสั่งที่อยู่ในเหตุการณ์ Load อย่างไรก็ตาม ก็มีบางจุดที่ผู้อ่านสามารถปรับปรุงให้โค้ดมีประสิทธิภาพได้ ด้วยเทคนิคต่อไปนี้

- หลีกเลี่ยงการใช้ตัวแปรชนิด Variant
- พยายามเลือกใช้ตัวแปรและทำการคำนวณแบบ Integer
- หลีกเลี่ยงการเรียกใช้พรอพเพอร์ตี้ส์ของคอนโทรลโดยตรงด้วยการเก็บค่าที่จะใช้ไว้ในตัวแปรแทน

ปกติการใช้ตัวแปรในวิชวลเบสิกนั้น หากผู้ใช้ไม่ประกาศชนิดแน่นอนวิชวลเบสิกจะถือว่าเป็นชนิด Variant ทันที ผู้ใช้สามารถปรับปรุงการเขียนโปรแกรมโดยหลีกเลี่ยงการใช้ตัวแปร Variant แต่ในวิชวลเบสิก 4.0 จะใช้ขั้นตอนสำหรับจัดการตัวแปรชนิด Variant ได้เร็วกว่าวิชวลเบสิก 3.0 ทางที่ดีที่สุดผู้อ่านควรใช้ Option Explicit เพื่อเป็นการบังคับให้ผู้ใช้ต้องประกาศตัวแปรก่อนใช้งานเสมอ

เมื่อต้องทำการคำนวณ ผู้อ่านควรใช้ตัวแปรชนิด Byte, Integer และ Long มากกว่าที่จะใช้ชนิด Currency, Single และ Double เพราะชนิดตัวแปรเหล่านั้นเป็นชนิดพื้นฐานของซีพียูจึงทำให้โอเปอเรชันทำได้เร็วกว่า แต่กรณีที่จำเป็นต้องใช้เลขจำนวนจริงในการคำนวณก็ให้ใช้ Double แทน Currency เพราะจะคำนวณได้เร็วกว่าโดยเฉพาะบนเครื่องที่มีแมธโคโปรเซสเซอร์ ซึ่งมักมีบนเครื่องที่ใช้งานวินโดวส์ 95 หรือเอ็นทีอยู่

การใช้ตัวแปรแทนการเรียกใช้พรอพเพอร์ตี้ส์โดยตรงจะช่วยให้การทำงานของโปรแกรมเร็วขึ้น 10 - 20 เท่า โดยเฉพาะกรณีที่ผู้เขียนโปรแกรมรู้ว่าค่าของพรอพเพอร์ตี้ส์นั้นไม่มีการเปลี่ยนแปลงค่าแน่นอน ดังนั้นจึงควรหลีกเลี่ยง เช่น

```
For i = 0 To 10
picIcon(i).Left = picPallete.Left
Next i
```

ควรแก้ไขใหม่เป็น

```
picLeft = picPallete.Left
For i = 0 To 10
picIcon(i).Left = picLeft
Next i
```

เช่นเดียวกับโค้ดต่อไปนี้

```
Do Until EOF(F)
Line Input #F, nextLine
Text1.Text = Text1.Text + nextLine
Loop
```

จะทำงานช้ากว่าโค้ดต่อไปนี้

```
Do Until EOF(F)
Line Input #F, nextLine
bufferVar = bufferVar + nextLine
Loop
Text1.Text = bufferVar
```

แต่จริง ๆ แล้วเราสามารถเขียนโค้ดในบรรทัดต่อไปนี้เพื่อแทนการทำงานข้างต้นทั้งหมด

```
Text1.Text = Input(F, LOF(F))
```

แต่การเขียนโปรแกรมด้วยชุดคำสั่งจะช่วยให้ทำความเข้าใจได้ง่ายกว่า

ในวิชวลเบสิก 4.0 จะเพิ่มความสามารถให้วิชวลเบสิกฟอร์แอพพลิเคชันเรื่องการใช้คอลเลคชันของออบเจ็กต์ แต่ต้องเลือกใช้ให้ถูกต้องดังนี้

- เลือกใช้ประโยค For Each...Next แทนการใช้ For(index)...Next
- ไม่ควรใช้พารามิเตอร์ before และ after ในกรณีที่จะเพิ่มออบเจ็กต์ในคอลเลคชัน
- ใช้คีย์ของคอลเลคชันแทนการใช้อินเด็กซ์อะเรย์สำหรับกรุ๊ปของออบเจ็กต์ชนิดเดียวกัน

ผู้อ่านสามารถเขียนโปรแกรมเพื่อเข้าถึงออบเจ็กต์ของคอลเลคชันได้โดยใช้ For...Next แต่ควรใช้ For Each...Next แทนเพราะทำงานได้เร็วกว่า แต่ความเร็วจะขึ้นกับแต่ละคอลเลคชันที่สร้างด้วย นอกจากนี้เมื่อใช้ Add เพิ่มออบเจ็กต์ ก็ไม่ควรใช้พารามิเตอร์ before และ after เพราะเป็นการสั่งให้วิชวลเบสิกต้องค้นหาออบเจ็กต์ที่ต้องการก่อน ทำให้เสียเวลาไปเปล่า ๆ ดังนั้นก่อนที่จะใช้คอลเลคชันก็ควรวางแผนให้ดีเพราะหากเรากำหนดคีย์ไว้กับออบเจ็กต์แต่ละอัน การใช้ออบเจ็กต์ผ่านคีย์จะทำได้เร็วกว่าใช้แบบอะเรย์มาก

ปรับปรุง Display Speed

เนื่องจากวินโดวส์ทำงานในโหมดกราฟิก ดังนั้นความเร็วในการจัดการกราฟิกก็จะมีผลต่อการทำงานของโปรแกรมด้วย ผู้อ่านสามารถปรับให้การทำงานของแอพพลิเคชันเร็วขึ้น โดยปรับที่ให้การระบายสีซ้ำ (Repaint) ของฟอร์มเร็วขึ้นได้ดังนี้

- กำหนดให้พรอพเพอร์ตี้ส์ ClipControls เป็น False
- กำหนดค่าพรอพเพอร์ตี้ส์ AutoRedraw ให้เหมาะสม
- เลือกใช้คอนโทรล Image แทนคอนโทรล PictureBox
- ถ้าเป็นไปได้เลือกใช้เมธอด Line แทน PSet
- พยายามซ่อนคอนโทรลเมื่อมีการกำหนดค่าพรอพเพอร์ตี้ส์เพื่อป้องกันการระบายซ้ำ

หากในโปรแกรมไม่จำเป็นต้องใช้เมธอดที่เกี่ยวกับกราฟิก เช่น Line, PSet, Circle, Print ก็ควรกำหนดพรอพเพอร์ตี้ส์ ClipControls ของฟอร์ม คอนโทรลเฟรม หรือพิกเจอร์บ็อกซ์ให้เป็น False เพราะจะช่วยให้วิชวลเบสิกไม่ต้องทำงานเพิ่ม เพื่อป้องกันการวาดทับบนคอนโทรลตอนระบายซ้ำ ดังนั้นบนฟอร์มที่มีคอนโทรลมาก ๆ จะเห็นว่าความเร็วของโปรแกรมจะเพิ่มขึ้นพอสมควร

การปรับค่าพรอพเพอร์ตี้ส์ AutoRedraw จะขึ้นอยู่กับสิ่งที่กำลังแสดงอยู่ ผู้อ่านสามารถเพิ่มความเร็วการวาดซ้ำของส่วนประกอบต่าง ๆ บนฟอร์มหรือคอนโทรลพิกเจอร์โดยกำหนดพรอพเพอร์ตี้ส์ AutoRedraw ให้เป็น False แล้วจัดการด้วยกราฟิกเมธอดในเหตุการณ์ Paint แทน แต่กรณีที่กราฟิกนั้นซับซ้อนเกินไปก็ควรกำหนด AutoRedraw ให้เป็น True เพื่อให้วิชวลเบสิกจัดการเรื่องนี้ให้คุณแทน โดยวิชวลเบสิกจะใช้หน่วยความจำบางส่วนมาเก็บบิตแม็พนั้นแทน

หากคุณไม่จำเป็นต้องใช้คุณสมบัติพิเศษของคอนโทรล PictureBox เช่น DDE, กราฟิกเมธอดแล้ว ก็ควรใช้คอนโทรล Image แทน เพราะการทำงานของคอนโทรลเร็วกว่ามาก

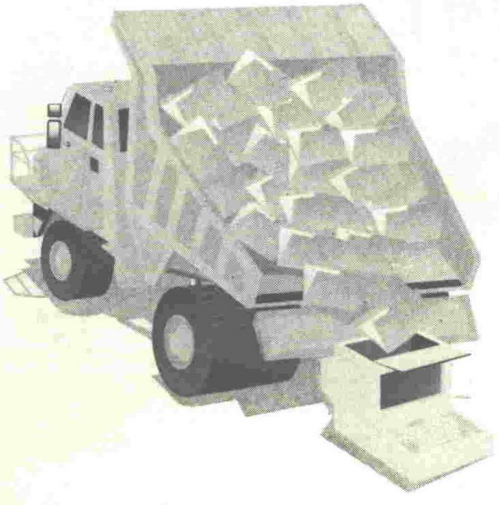
การกำหนดตำแหน่งเพื่อทำกราฟิกด้วยเมธอด Line พบว่าทำได้เร็วกว่าการใช้เมธอด PSet ซ้ำ ๆ กัน ดังนั้นควรหลีกเลี่ยงการใช้ PSet โดยเปลี่ยนมาใช้เมธอด Line แทน

กรณีที่มีการแก้ไขพรอพเพอร์ตี้ส์ของคอนโทรล วิชวลเบสิกจะต้องระบายสีคอนโทรลนั้นซ้ำให้เสมอ การระบายแต่ละครั้งจะเสียเวลามากพอสมควร ดังนั้นก่อนที่จะแก้ไขพรอพเพอร์ตี้ส์ของคอนโทรลก็ควรจะซ่อนคอนโทรลนั้นก่อน เช่นกรณีที่เราจะแก้ไขขนาดของคอนโทรลลิสต์บ็อกซ์ทุกครั้งเมื่อมีการปรับขนาดฟอร์ม

```
Sub Form_Size ()
Dim I As Integer, sHeight As Integer
sHeight = ScaleHeight / 4
For I = 0 To 3
lstDisplay(I).Move 0,I *sHeight,ScaleWidth, sHeight
Next
End Sub
```

หากผู้อ่านเขียนโค้ดตามที่แสดงข้างต้นจะทำให้เกิดการระบายซ้ำถึง 4 ครั้งสำหรับลิสต์บ็อกซ์แต่ละอัน ซึ่งมีวิธีแก้ไขโดยนำลิสต์บ็อกซ์ทั้ง 4 อัน ไปใส่ไว้ในคอนโทรล PictureBox แล้วซ่อนพิกเจอร์บ็อกซ์ก่อนที่จะเคลื่อนย้าย และเมื่อทำเสร็จจึงแสดงพิกเจอร์บ็อกซ์ใหม่ เพื่อทำให้เกิดการระบายซ้ำครั้งเดียวดังนี้

```
Sub Form_Resize ()
Dim I As Integer, sHeight As Integer
picContainer.Visible = False
picContainer.Move 0, 0, ScaleWidth, ScaleHeight
sHeight = ScaleHeight / 4
For I = 0 To 3
lstDisplay(I).Move 0,I* sHeight,ScaleWidth, sHeight
Next
picContainer.Visible = True
End Sub
```



ปรับปรุง Apparent Speed

ผู้ใช้จะรู้สึกว่าแอพพลิเคชันทำงานได้เร็วเมื่อแอพพลิเคชันปรากฏตัวขึ้นทันทีหลังจากเรียกใช้ แทนที่จะต้องเสียเวลารอกว่าฟอร์มจะปรากฏขึ้น โดยทั่วไปก็มีหลักการบางอย่างที่สามารถนำมาใช้ให้รู้สึกว่าแอพพลิเคชันทำงานเร็วขึ้นดังนี้

- ซ่อนฟอร์มที่โหลดไว้ก่อนขณะที่ยังไม่ใช้
- โหลดข้อมูลที่จำเป็นต้องใช้ให้เสร็จในครั้งเดียว
- ใช้ไทม์เมอร์ทำงานบางอย่างในแบคกราวนด์
- ใช้คอนโทรล ProgressBar ประกอบ

ฟอร์มบางอันที่โหลดมาใช้งานแล้วไม่จำเป็นต้องอันโหลดทุกครั้ง เราสามารถซ่อนฟอร์มไว้ในช่วงที่ไม่ได้ใช้งาน เพราะหากเราต้องการใช้งานก็สามารถแสดงได้ทันที ในกรณีนี้แอพพลิเคชันจะกินเนื้อที่หน่วยความจำเพิ่มขึ้น แต่ก็แอพพลิเคชันก็ทำงานเร็วขึ้น กรณีที่แอพพลิเคชันจำเป็นต้องใช้ข้อมูลบางอย่าง เราควรจะโหลดข้อมูลเหล่านั้นให้เสร็จภายในครั้งเดียวเลย นอกจากนี้ควรนำคอนโทรลไทม์เมอร์มาใช้ประโยชน์ เพราะเวลาส่วนหนึ่งที่เสียไปของแอพพลิเคชันใช้รอการตอบสนองจากผู้ใช้โปรแกรม ดังนั้นเราจึงสามารถนำงานบางอย่างไปทำในคอนโทรลไทม์เมอร์ในช่วงเวลาที่ว่างได้ แต่มีข้อจำกัดที่งานชิ้นนั้นต้องไม่ใหญ่เกินไป อย่างไรก็ตาม หากไม่สามารถหลีกเลี่ยงงานที่ต้องทำขณะโหลดแอพพลิเคชันก็ควรนำคอนโทรล ProgressBar ซึ่งมีในวิชวลเบสิก 4.0 รุ่น 32 บิต เพื่อใช้แสดงความก้าวหน้าของโปรแกรมและบอกให้ผู้ใช้ทราบว่าขณะนั้นโปรแกรมยังทำงานอยู่ นอกจากนี้ก็ควรเรียกฟังก์ชัน DoEvents ในแต่ละช่วงที่ปรับค่าของ ProgressBar เพื่อให้แอพพลิเคชันระบายซ้ำและทำงานอื่นได้ด้วย

สิ่งที่หลีกเลี่ยงไม่ได้ขณะโหลดแอพพลิเคชันวิชวลเบสิกจำเป็นต้องโหลดไดนามิกลิงค์ไลบรารี (DLLs) ที่จำเป็นเช่น VBA, OLE แต่ก็มีเทคนิคเล็ก ๆ น้อย ๆ ที่ทำให้รู้สึกว่าแอพพลิเคชันโหลดเร็วขึ้น คือ

- ใช้เมธอด Show ช่วยในเหตุการณ์ Load
- สร้างฟอร์มเล็ก ๆ สำหรับทำหน้าที่ตอนแอพพลิเคชันเริ่มต้นทำงานเท่านั้น
- หลีกเลี่ยงการโหลดโมดูลที่ไม่จำเป็น
- เรียกวิชวลเบสิกแอพพลิเคชันเล็ก ๆ มาทำงานก่อน เพื่อใช้โหลด DLLs

ขณะที่ฟอร์มกำลังโหลดอยู่นั้นโค้ดทั้งหมดจะถูกทำงานก่อนที่ฟอร์มจะแสดงขึ้นมา ดังนั้นเราสามารถแก้ไขได้โดยใช้เมธอด Show ในเหตุการณ์ Load ก่อนที่จะทำงานอื่นต่อไป และที่สำคัญต้องเรียก DoEvents ด้วยเพื่อให้แน่ใจว่าฟอร์มถูกระบายสีเรียบร้อย เช่น

```
Sub Form_Load ()
Show                    'Display startup form
DoEvents                'Ensure startup form is painted
Load MainForm 'Load main application form
Unload Me               'Unlaod startup form
MainForm.Show           'Display main form
End Sub
```

สำหรับฟอร์มที่มีคอนโทรลมากและซับซ้อน ผู้ใช้ควรสร้างฟอร์มเล็ก ๆ ที่มีคอนโทรลไม่ซับซ้อนมาทำหน้าที่โหลดแอพพลิเคชันโดยเฉพาะ แอพพลิเคชันของไมโครซอฟท์หลายตัวที่ใช้วิธีนี้ คือสร้างฟอร์มเล็ก ๆ ที่แสดงชื่อและลิขสิทธิ์ของแอพพลิเคชันมาให้ผู้ใช้เห็นก่อนระหว่างที่กำลังโหลด ส่วนของแอพพลิเคชันจริง ๆ จากตัวอย่างโค้ดข้างต้นนี้เราสร้างฟอร์มเล็ก ๆ สมมติชื่อ Startup มาเพื่อโหลดฟอร์มของแอพพลิเคชัน MainForm โดยเราบังคับให้ฟอร์ม Startup แสดงตัวทันทีด้วยเมธอด Show แล้วจึงโหลด MainForm จากนั้นอันโหลดฟอร์ม Startup แล้วแสดงฟอร์ม MainForm ขึ้นมา

ปรับขนาดหน่วยความจำ

มีหลายวิธีที่เราสามารถลดขนาดการใช้หน่วยความจำของแอพพลิเคชันโดยใช้เทคนิคต่อไปนี้

- เรียกหน่วยความจำที่ใช้โดยตัวแปรสตริง อะเรย์ และออบเจ็กต์ที่ไม่ได้ใช้แล้วคืน
- หลีกเลี่ยงการใช้ตัวแปรชนิด Variant
- หลีกเลี่ยงการใช้ตัวแปรสตริงแบบความยาวคงที่
- ตรวจงานโค้ดที่ไม่เคยถูกเรียกใช้แล้วลบออกจากโปรแกรม

ตัวแปรสตริงหรืออะเรย์ที่ประกาศใช้อยู่แบบโกลบอลหรือที่อยู่ในโมดูลต่าง ๆ จะใช้เนื้อที่บนหน่วยความจำตลอดเวลาที่แอพพลิเคชันกำลังทำงานอยู่ ดังนั้นหากตัวแปรใดที่ไม่จำเป็นต้องใช้ก็ควรจะเรียกหน่วยความจำคืน สำหรับตัวแปรสตริงทำได้โดยกำหนดความยาวของสตริงให้เป็นศูนย์ เช่น

```
myString = ""
```

ส่วนตัวแปรไดนามิกอะเรย์จะเรียกคืนด้วยประโยค Erase หรือกำหนดขนาดใหม่ด้วยประโยค ReDim Preserve เช่น

```
Erase bigArray
ReDim Preserve bigArray(10, smallerNum)
```

กรณีตัวแปรที่เป็นออบเจ็กต์ให้กำหนดตัวแปรเป็น Nothing เช่น

```
Global F As New StatusForm
F.Show 1                'Form is loaded and show modally
X = F.Text1.Text        'User presses a button that hides form
UnLoad F                'Get rid of visual part of form
Set F = Nothing         'Recalim space of module data
```

ตัวแปรชนิด Variant แต่ละตัวต้องใช้เนื้อที่ 16 ไบต์ เทียบกับชนิด Integer ที่ใช้เพียง 2 ไบต์ หรือ Double ซึ่งใช้ 8 ไบต์ สำหรับตัวแปรชนิดสตริงแบบความยาวไม่คงที่จะใช้เนื้อที่ 4 ไบต์ เพื่อเก็บพอยน์เตอร์ บวกค่าความยาวของสตริงและอีก 2 ไบต์ สำหรับเก็บคาแรคเตอร์ Null terminating และในการส่งผ่านอาร์กิวเมนต์แบบอ้างอิงต้องใช้เนื้อที่ 4 ไบต์ ดังนั้นการส่งผ่านข้อมูลชนิด Variant ที่เก็บสตริงแบบอ้างอิงจะต้องใช้เนื้อที่ทั้งหมด 22 ไบต์ บวกความยาวของสตริง ดังนั้นหากใช้วิชวลเบสิกรุ่น 16 บิต อาจเกิดปัญหาสแต็กมีเนื้อที่ไม่พอ แต่ถ้าใช้รุ่น 32 บิต ก็จะไม่ค่อยมีปัญหา

กรณีสุดท้ายในโปรแกรมที่ยาว ๆ บางครั้งอาจมีโค้ดหรือตัวแปรบางตัวที่ใช้สำหรับทดสอบโปรแกรมซ่อนอยู่ได้ ดังนั้นเมื่อเขียนโปรแกรมเสร็จ ผู้เขียนโปรแกรมก็ควรมองหาโพรซีเดอร์หรือตัวแปรเหล่านี้แล้วจัดการลบออกให้หมด เพราะสิ่งเหล่านี้จะกินเนื้อที่หน่วยความจำได้

ปรับปรุงการใช้เนื้อที่สำหรับกราฟิก

กราฟิกในแอพพลิเคชันก็เป็นการใช้เนื้อที่บนหน่วยความจำอีกส่วนหนึ่ง ซึ่งมีเทคนิคที่ใช้จัดการกราฟิกเหล่านี้ให้มีประสิทธิภาพที่ดีดังนี้

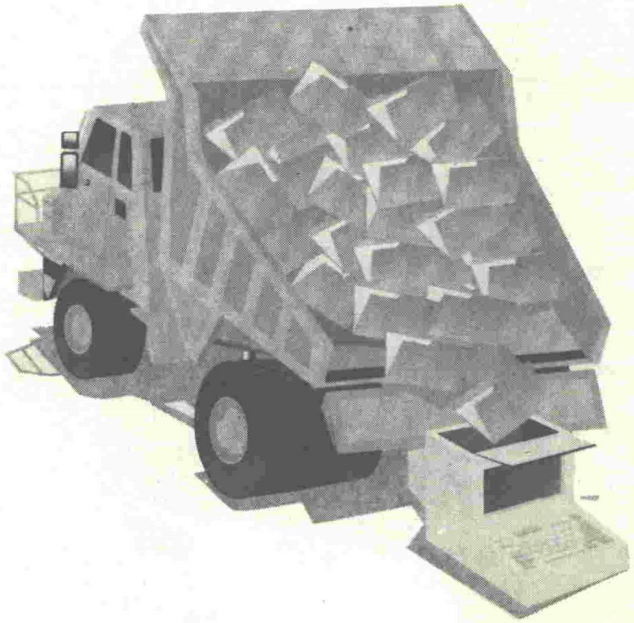
- เรียกเนื้อที่กราฟิกคืนด้วยฟังก์ชัน LoadPicture() และเมธอด Cls
- ใช้คอนโทรล Image แทน PictureBox
- โหลดภาพเท่าที่จำเป็น นอกจากนี้ก็ควรใช้ภาพร่วมกัน
- พยายามเลือกใช้ไฟล์ RLE หรือเมตาไฟล์แทน

สำหรับกราฟิกใน PictureBox หรือ Image ที่ไม่ต้องใช้อีกแล้วควรเรียกเนื้อที่คืนด้วย LoadPicture() และ Cls แทนการซ่อนไว้เฉย ๆ ดังนี้

```
Image1.Picture = LoadPicture()
```

สำหรับวิชวลเบสิก 4.0 เราสามารถใช้วิธีนี้สำหรับเรียกเนื้อที่คืนแทน

```
Set Image1.Picture = Nothing
```



นอกจากนี้ยังอาจใช้พรอพเพอร์ตี้ส์ AutoRedraw มาช่วยในการเรียกเนื้อที่คืนจากบิตแม็พบนฟอร์มหรือพิกเจอร์คอนโทรลดังนี้

```
myPic.AutoRedraw = True
myPic.Cls
myPic.AutoRedraw = False
```

การใช้ Image จะทำงานได้เร็วกว่า PictureBox 5 ถึง 10 เท่า ดังนั้นหากไม่จำเป็นต้องใช้คุณสมบัติพิเศษของพิกเจอร์บ็อกซ์ เช่น แคร็กแอนด์ดร็อบ, DDE, กราฟิกเมธอด หรือใช้เป็นตัวบรรจุคอนโทรลอื่น นอกจากนี้ไม่ควรเก็บภาพไว้ในแอพพลิเคชันขณะที่ออกแบบ แต่ควรโหลดกราฟิกมาใช้ระหว่างที่โปรแกรมกำลังทำงานเพื่อประหยัดการใช้หน่วยความจำ และถ้าจะใช้ภาพเดียวกันในคอนโทรลพิกเจอร์บ็อกซ์หรืออิมเมจหลาย ๆ อันก็ไม่ควรเขียนดังนี้

```
Picture = LoadPicture("c:\windows\chess.bmp")
Image1.Picture = LoadPicture("c:\windows\chess.bmp")
Picture1.Picture = LoadPicture("c:\windows\chess.bmp")
```

เพราะจะมีภาพเดียวกัน 3 ภาพ เก็บบนหน่วยความจำ ซึ่งเปลืองเนื้อที่และเสียเวลาเปล่า ๆ ควรเปลี่ยนเป็น

```
Picture =LoadPicture("c:\windows\chess.bmp")
Image1.Picture = Picture("c:\windows\chess.bmp")
Picture1.Picture = Picture("c:\windows\chess.bmp")
```

เช่นเดียวกับการใช้ภาพเดียวกันในหลาย ๆ ฟอร์ม ถ้าแต่ละฟอร์มต่างก็โหลดภาพเองก็จะเปลืองที่ ดังนั้นควรเก็บกราฟิกแยกไว้ในส่วนรีซอร์สของโปรแกรม และเลือกโหลดรูปที่ต้องการขณะโปรแกรมทำงานด้วยฟังก์ชัน LoadResPicture แทน หรือกรณีที่จะใช้บิตแม็พวาดเป็นพื้นบนฟอร์ม ก็ควรใช้เมธอด PaintPicture แทนการใช้คอนโทรล เพราะสามารถใช้ PaintPicture วาดซ้ำได้โดยไม่จำกัด และถ้าเป็นไปได้ควรใช้กราฟิกไฟล์ฟอร์แมต Run Length Encoded (.RLE) หรือเมตาไฟล์ก่อน เพราะขนาดเล็กกว่ามาก

แยกส่วนแอพพลิเคชันด้วย OLE

วิชวลเบสิก 4.0 ได้เพิ่มลักษณะการใช้ OLE ให้มีประสิทธิภาพมากขึ้น แทนที่จะเขียนโปรแกรมทั้งหมดไว้ในไฟล์ ๆ เดียว คุณสามารถแยกโปรแกรมออกเป็น OLE servers หลาย ๆ ส่วน ประโยชน์ที่ได้จากการนำ OLE มาใช้ก็คือ

- OLE servers จะถูกโหลดมาเมื่อต้องการใช้ และอันโหลดได้ถ้าไม่ใช้

- OLE servers แบบ Out-of-process ทำงานแบบ 32 บิตบนวินโดวส์ 95 หรือเอ็นที สามารถเรียกใช้จากแอพพลิเคชันแบบ 16 บิตได้

- Remote OLE servers สามารถใช้รีซอร์สของคอมพิวเตอร์เครื่องอื่นบนเน็ตเวอร์กได้

นอกจากนี้ OLE servers แต่ละตัว สามารถแยกกันดีบักเมื่อมีข้อผิดพลาด และนำกลับมาใช้ซ้ำได้ ช่วยให้ประหยัดเวลาไม่ต้องเขียนโปรแกรมส่วนนั้นขึ้นมาใหม่อีก

OLE servers แบบ Out-of-process จะเป็นโปรแกรมที่ทำงานได้เองตัวหนึ่งที่ให้บริการโอเล่กับโปรแกรมอื่น ๆ งานที่ทำของโปรแกรมจะใช้เนื้อที่สแต็กของตัวเอง เมื่อใดก็ตามที่ไคล์เอ็นต์ใช้โอเล่ออบเจ็กต์ โปรเซสจะข้ามจากไคล์เอ็นต์ไปยังเซอร์ฟเวอร์ จุดเด่นของ Out-of process ซึ่งมีเหนือชนิดอื่นก็คือ ทำงานแบบอะซิงโครนัส การแสดงฟอร์มเป็นแบบนอนโมเดล ป้องกัน Error ที่จะเกิดในเซอร์ฟเวอร์ซึ่งอาจทำให้โปรแกรมหยุดทำงานได้ ติดต่อกับโปรแกรมได้ทุกชนิดไม่ว่าจะเป็น 16 หรือ 32 บิต แต่ Out-of-process ก็มีข้อเสียที่เหมือนโปรแกรมที่เขียนด้วยวิชวลเบสิกก็คือ ความเร็วตอนเริ่มต้นจะต้องโหลด DLLs หลาย ๆ ตัว และระหว่างการทำงานจะต้องใช้ซีพียูไซเคิลหลายรอบทำให้การใช้งานแบบนี้จะช้ากว่า In-of-process

OLE servers แบบ In-of-process จะเป็น DLL ที่ให้บริการโอเล่กับโปรแกรม การทำงานจะใช้สแต็กและเนื้อที่ของไคล์เอ็นต์ตลอดจึงไม่ต้องเสียเวลาข้ามไปข้ามมาระหว่างเซอร์ฟเวอร์กับไคล์เอ็นต์ การทำงานจึงเร็วกว่า Out-of-process สำหรับ In-of-process จะต้องทำงานแบบ 32 บิตเท่านั้น และไม่สามารถใช้แบบโมเดลฟอร์มได้

Remote OLE servers ในวิชวลเบสิก 4.0 รุ่นเอ็นเตอร์ไพซ์สามารถสร้างเซอร์ฟเวอร์ชนิดนี้ได้ เพื่อให้ทำงานบนเครื่องอื่นบนเน็ตเวอร์ก ข้อมูลจะถูกส่งผ่านกันบนเน็ตเวอร์ก

ปรับปรุงการใช้ OLE

การเรียกใช้โอเล่ออบเจ็กต์ภายในแอพพลิเคชันที่เขียนด้วยวิชวลเบสิกนั้นมีเทคนิคบางอย่างที่จะทำให้ออบเจ็กต์ทำงานได้อย่างมีประสิทธิภาพดังนี้

- ใช้ Early binding แทน Late binding
- พยายามใช้ดอทโอเปเรเตอร์ให้น้อยที่สุด
- ใช้ Set และ With...End With ช่วย
- พยายามใช้ Out-of-process ให้น้อยที่สุด

การใช้ออบเจ็กต์ในวิชวลเบสิกให้มีประสิทธิภาพถ้าสามารถทำ binding ได้เร็วตั้งแต่ต้นถ้าผู้ใช้อ้างถึงไทป์ไลบรารีที่บรรจุออบเจ็กต์นั้น เช่น

```
Dim X As New MyObject
```

หรือจะเขียนใหม่ดังนี้

```
Dim X As MyObject
Set X = New MyObject
```

หากทำ Early binding ได้วิชวลเบสิกจะสามารถจัดการออบเจ็กต์ได้ตั้งแต่ตอนคอมพายล์ แทนที่จะไปเสียเวลาทำตอนโปรแกรมทำงาน นอกจากนี้ยังช่วยตรวจสอบการใช้พรอพเพอร์ตี้ส์และเมธอด หากมีข้อผิดพลาดเกิดขึ้นก็จะทราบได้ทันที

การอ้างอิงถึงโอเล่ออบเจ็กต์จะต้องใช้ดอท (".") ช่วยโดยอ้างจากคอลเลคชัน ออบเจ็กต์ พรอพเพอร์ตี้ส์ และเมธอด เช่น

```
Application.Workbooks.Item(1).Worksheets.
Item("Sheet1").Cells.Item(1,1)
```

เป็นการอ้างถึงเซล A1 บนชีต Sheet1 ในเวอร์กบุ๊ค จะเห็นว่าดูซับซ้อนอ่านยาก การเรียกใช้โอเล่เซอร์ฟเวอร์ออบเจ็กต์ และการใช้ดอทแต่ละครั้งจะทำให้วิชวลเบสิกต้องเรียกไปยังโอเล่บ่อยขึ้น ดังนั้นควรพยายามลดการใช้ดอทให้น้อยที่สุด เช่น ในประโยคข้างต้นสามารถแก้ไขให้สั้นลงได้โดยตัด Item ออกเพราะเป็นเมธอดโดยปริยายเมื่ออ้างถึงคอลเลคชันและเปลี่ยนไปใช้เมธอด Range แทนดังนี้

```
Application.Workbooks(1)
.Worksheets("Sheet1").Range("A1")
```

หรือจะให้สั้นกว่านี้โดยพยายามอ้างถึงชีตและเวอร์กบุ๊คที่เอคทีฟแทน ทำให้เราอ้างเพียงดังนี้

```
Range("A1")
```

การใช้ประโยค Set จะช่วยให้เขียนโค้ดได้สั้นลงดังนี้

```
Dim xlRange As Object
Set xlRange = Application.ActiveSheet
.Cells(1,1)
xlRange.Font.Bold = True
xlRange.Width = 40
```

หรือจะใช้ With...End With แทน

```
With Application.ActiveSheet.Cells(1,1)
      Font.Bold = True
      Width = 40
End With
```

ในการใช้งานโอเล่เซอร์ฟเวอร์แบบ Out-of-process นั้นพยายามเรียกใช้ให้น้อยที่สุด พยายามใช้ตัวแปรและนำมาใช้ในลูปแทนที่จะเรียกใช้ค่าจากพรอพเพอร์ตี้ส์โดยตรง

สำหรับเทคนิคต่าง ๆ ที่กล่าวมานี้ หากผู้อ่านนำไปใช้ในการเขียนโปรแกรมแล้ว ก็จะสามารถทำให้โปรแกรมที่เขียนทำงานได้มีประสิทธิภาพมากขึ้น

เลหลังเครื่องเก่าจ้า !!

หากจะประกาศขายเครื่องคอมพิวเตอร์ที่ค้างสต๊อกอยู่ราคาถูก ๆ เครื่องละ 1,000 บาท มีใครสนใจบ้างหรือเปล่า สภาพยังใหม่แกะกล่องเพียงแต่ผลิตในปี ค.ศ. 1982 แค่สิบกว่าปีที่ผ่านมาเท่านั้นเอง

อย่าเพิ่งตกใจค่ะ ว่าจะยังมีเครื่องคอมพิวเตอร์อะไรจะเก่าขนาดนั้นเหลืออยู่ (ซึ่งที่จริงแล้วอาจจะมีเหลือจริง ๆ ก็ได้) จากการสำรวจของนิตยสาร Byte ในปี ค.ศ. 1982 หรือ พ.ศ. 2525 ได้สำรวจคลังเก็บคอมพิวเตอร์จำนวน 165 คลัง ปรากฏว่ามีเครื่องคอมพิวเตอร์อยู่ 37 ยี่ห้อที่สำรวจพบและในนั้นมี 14 ยี่ห้อที่มีจำนวนเครื่องคอมพิวเตอร์ขายไม่ออกเยอะที่สุด เรียงจากมากไปหาน้อยแล้ว ดังต่อไปนี้ค่ะ

ยี่ห้อ	ยอดคงเหลือ (เปอร์เซ็นต์)
Apple	73%
Atari	37%
IBM	23%
Commodore	22%
Vector Graphic	21%
Radio shack	21%
Cromemco	18%
Texas Tnstruments	13%
North Star	11%
Dynabyte	11%
Data General	10%
DEC	9%
Zenith	9%
Altos	9%